AFRL-OSR-VA-TR-2013-0571

# SEMI-SUPERVISED DISCRIMINATIVE STRUCTURED PREDICTION

**SHAOJUN WANG**

**WRIGHT STATE UNIVERSITY**

**10/30/2013**
**Final Report**

**AIR FORCE RESEARCH LABORATORY**
**AF OFFICE OF SCIENTIFIC RESEARCH (AFOSR)/RSL**
**ARLINGTON, VIRGINIA 22203**
**AIR FORCE MATERIEL COMMAND**

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| **1. REPORT DATE** *(DD-MM-YYYY)* 28-10-2013 | **2. REPORT TYPE** Final | **3. DATES COVERED** *(From - To)* August 1, 2010 - July 31, 2013 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** Semi-supervised Discriminative Structured Prediction | **5a. CONTRACT NUMBER** |
| | **5b. GRANT NUMBER** FA9550-10-1-0335 |
| | **5c. PROGRAM ELEMENT NUMBER** |
| **6. AUTHOR(S)** Shaojun Wang | **5d. PROJECT NUMBER** |
| | **5e. TASK NUMBER** |
| | **5f. WORK UNIT NUMBER** |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Wright State University | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)** AFOSR | **10. SPONSOR/MONITOR'S ACRONYM(S)** |
| | **11. SPONSOR/MONITOR'S REPORT NUMBER(S)** |

**12. DISTRIBUTION / AVAILABILITY STATEMENT**
A

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

The proposed research develops cutting-edge machine learning techniques to improve the performance of a wide spectrum of robust and intelligent classification tasks. Structured prediction, one of major challenges in statistical machine learning, is a classification or regression problem with non-iid data where the prediction variables are typically interdependent in complex ways with dependencies encoded in a graphical model to capture the sequential, spatial, relational or recursive structure of output variables. Semi-supervised learning, another example of the four major challenges in statistical machine learning, is a technique which makes use of both unlabeled and labeled data for training |typically a small amount of labeled data with a large amount of unlabeled data. Traditinal approaches optimize surrogate functions of performance measures for structured prediction. In this project, we propose to design novel machine learning algorithms that directly optimize performance measures for classification and ranking problems.

**15. SUBJECT TERMS**

| | | | | | |
|---|---|---|---|---|---|
| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | | **19b. TELEPHONE NUMBER** *(include area code)* |

# Final AFOSR Project Performance Report

Shaojun Wang
(Principal Investigator)
Wright State University

October 2013

**Abstract**

This PI was awarded the AFOSR grant "Semi-supervised Discriminative Structured Prediction" (Grant No.: FA9550-10-1-0335). The project was funded for the period of 08/01/10 to 07/31/13 with the total amount of $359,320. This report summarizes the progress made throughout the project period.

## 1 Research Progress

The proposed research develops cutting-edge machine learning techniques to improve the performance of a wide spectrum of robust and intelligent classification tasks. Structured prediction, one of four major challenges in statistical machine learning, is a classification or regression problem with non-iid data where the prediction variables are typically interdependent in complex ways with dependencies encoded in a graphical model to capture the sequential, spatial, relational or recursive structure of output variables. Semi-supervised learning, another example of the four major challenges in statistical machine learning, is a technique which makes use of both unlabeled and labeled data for training — typically a small amount of labeled data with a large amount of unlabeled data. Traditinal approaches optimize surrogate functions of performance measures for structured prediction. In this project, we propose to design novel machine learning algorithms that directly optimize performance measures for classification and ranking problems and maximize various arbitrarily defined margins with the goal to improve the generalization performance.

Consistent with the stated objectives of the project, the project has made considerable progress along the following four directions. First, we have proposed a boosting method that directly minimizes 0-1 loss and maximizes variously targeted arbitrarily defined margins for binary classification. Second, we have developed a semi-supervised boosting method that directly minimizes a combination of 0-1 loss over labeled examples and soft 0-1 loss over unlabeled examples, and maximizes various margins over both labeled and unlabeled examples where the margin of an unlabeled example is defined to be an expected soft margin. Third, we have proposed a boosting method that directly minimizes 0-1 loss and maximizes various margins for multiclass classification. Fourth, we have developed an optimization method for linear models and a boosting method that builds boosted trees to directly maximize performance measures for ranking.

The major findings along the above directions are described in more detail in the following four subsections.

### 1.1 Direct Boost for Binary Classification

Let $\mathcal{H} = \{h_1, ..., h_l\}$ denote the set of all possible weak classifiers that can be produced by the weak learning algorithm, where a weak classifier $h_j \in \mathcal{H}$ is a mapping from an instance space $\mathcal{X}$ to $\mathcal{Y} = \{-1, 1\}$. The $h_j$s are not assumed to be linearly independent, and $\mathcal{H}$ is closed under negation, i.e., both $h$ and $-h$ belong to $\mathcal{H}$. We define $\mathcal{C}$ of $\mathcal{H}$ as the set of mappings that can be generated by taking a weighted average of classifiers from $\mathcal{H}$:

$$\mathcal{C} = \left\{ f : x \rightarrow \sum_{h \in \mathcal{H}} \alpha_h h(x) \mid \alpha_h \geq 0 \right\}, \tag{1}$$

Given a set of training data $\mathcal{D} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ independently drawn from an unknown but fixed probability distribution $p(X, Y)$, we consider finding $f \in \mathcal{C}$ that minimizes the empirical classification error in (2) and has a good generalization performance.

$$\text{error}(f, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(\hat{y}_i \neq y_i) \tag{2}$$

where $\hat{y}_i = \arg\max_{y \in \mathcal{Y}} \; y f(x_i)$ and $\mathbf{1}(\cdot)$ is the classification error function, i.e., an indicator function. Due to the nonconvexity, nondifferentiability and discontinuity of the classification error function and the max operation for $\hat{y}_i$, direct minimization of (2) seems impossible. In the following, we describe novel methods that directly minimize (2) and maximize margins.

### 1.1.1 Minimizing 0-1 Loss

DirectBoost, we propose, works by sequentially running an iterative greedy coordinate descent algorithm, and each time directly minimizes the *true* classification error (2) instead of a weighted classification error in AdaBoost [6]. Consider the $t$th iteration, the ensembled classifier is $f_t(x) = \sum_{k=1}^{t} \alpha_k h_k(x)$, where previous $t-1$ weak classifiers $h_k(x)$ and corresponding weights $\alpha_k, k = 1, \cdots, t-1$ have been selected and determined. Denote $a(x_i) = \sum_{k=1}^{t-1} \alpha_k h_k(x_i)$, then the inference function for sample $x_i$ can be written as,

$$F_t(x_i, y) = y \; h_t(x_i)\alpha_t + y a(x_i) \tag{3}$$

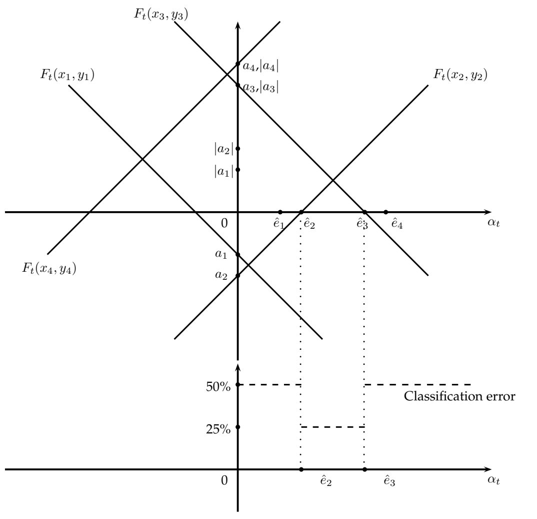

Figure 1: *An example of computing minimum 0-1 loss of a weak learner over 4 samples.*

We now describe the greedy coordinate descent algorithm that sequentially minimizes a 0-1 loss, please see the details in [25]. Since $\mathcal{H}$ is closed in negation, we only care about these that are positive. We first sort $|a(x_i)|, i = 1, \cdots, n$ in an increasing order. Then for a weak learner, we visit each sample in the order that $|a(x_i)|$ is increasing, and we compute the slope and the intercept of $F(x_i, y_i) = y_i h_k(x_i)\alpha + y_i a(x_i)$. Let $\hat{e}_j = |a(x_i)|$. If the slope is positive and $a(x_i)$ is positive, the sample margin is positive for $\alpha_t > 0$, thus there is no error update on the righthand side of $\hat{e}_j$; if the slope is positive and the intercept is negative, there is an error reduction on the righthand side of $\hat{e}_j$; if the slope is negative and the intercept is positive, there is an error increment on the righthand side of $\hat{e}_j$; if the slope is negative and the intercept is negative, the sample margin is always negative for $\alpha_t > 0$, thus there is no error update on the righthand side of $\hat{e}_j$. We incrementally calculate the classification error on intervals of $\hat{e}_j s$, and choose the interval with the minimum classification error. Consider an example with 4 samples. Suppose for a weak learner, we have $F_t(x_i, y_i), i = 1, 2, 3, 4$ as shown in Figure 3. At $\alpha_t = 0$, samples $x_1$ and $x_2$ have negative margins, thus they are misclassified, the error is 2 and the error rate is 50%. We incrementally update the classification error on intervals of $\hat{e}_i, i = 1, 2, 3, 4$: For $F_t(x_1, y_1)$, its slope is negative and its intercept is negative, sample $x_1$ always has a negative margin for $\alpha_t > 0$, thus there is no error update on the righthand side of $\hat{e}_1$. For $F_t(x_2, y_2)$, its slope is positive and its intercept is negative, then when $\alpha_t$ is at the right side of $\hat{e}_2$, sample $x_2$ has a positive margin and becomes correctly classified, so we update the error by -1, the error rate is reduced to 25%. For $F_t(x_3, y_3)$, its slope is negative and its intercept is positive, then when $\alpha_t$ is at the right side of $\hat{e}_3$, sample $x_3$ has a negative margin and becomes misclassified, so we update error by 1, the error rate changes to 50% again. For $F_t(x_4, y_4)$, its slope is positive and its intercept is positive, sample $x_4$ always has a positive margin for $\alpha_t > 0$, thus there is no error update on the righthand side of $\hat{e}_4$. We finally have the minimum error rate 25% on the interval of $[\hat{e}_2, \hat{e}_3]$.

We pick the weak learners, each having an interval with the largest classification error reduction. Since the classification error is flat on the interval with a minimum classification error, we determine the optimal weight of each selected weak learner by minimizing the exponential loss within the corresponding interval. We only add the weak learner with the smallest exponential loss into the ensembled classifier. We repeat this procedure until the training error reaches its minimum, which is 0 in a data separable case. We then go to the next stage, explained below, that aims to maximize margins. A nice property of the above greedy coordinate descent algorithm is that the classification error is monotonically decreasing, and its computational complexity is $\Theta(tMn)$ where $M$ is the number of weak learners considered by the weak learner algorithm at each round and is identical to the one in AdaBoost.

### 1.1.2 Maximizing Margins

The margins theory [15] provides an insightful analysis for the success of AdaBoost where the authors proved that the generalization error of any ensemble classifiers is bounded in terms of the entire distribution of margins of training examples, as well as the number of training examples and the complexity of the base classifiers, and AdaBoost's dynamics has a strong tendency to increase the margins of training examples. This view motivates us to prove that the generalization error of any ensemble classifiers is bounded in terms of the statistics of margins of training examples, as well as the number of training examples and the complexity of the base classifiers, and propose a coordinate ascent algorithm to directly maximize several types of margins just right after the training error reaches a (local) minimum.

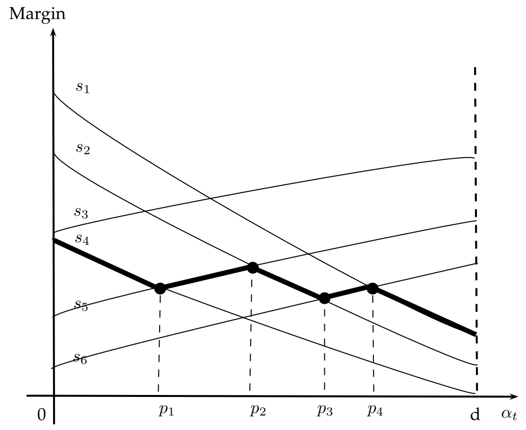

Figure 2: *Margin curves of six examples. At points $p_1, p_2, p_3$ and $p_4$, the median example is changed. At points $p_2$ and $p_4$, the set of bottom $n' = 3$ examples are changed.*

The margin of a labeled example $(x_i, y_i)$ with respect to an ensembled classifier $f_t(x) = \sum_{k=1}^{t} \alpha_k h_k(x_i)$ is defined to be

$$m_i = \frac{y_i \sum_{k=1}^{t} \alpha_k h_k(x_i)}{\sum_{k=1}^{t} \alpha_k} \tag{4}$$

We denote $a_i = \sum_{k=1}^{t-1} y_i \alpha_k h_k(x_i)$, $b_{i,t} = y_i h_t(x_i) \in \{-1, +1\}$ and $c = \sum_{k=1}^{t-1} \alpha_k$, then the margin on the $i$th example $(x_i, y_i)$ can be rewritten as $m_i = \frac{a_i + b_{i,t}\alpha_t}{c + \alpha_t}$. The derivative of the margin on $i$th example with respect to $\alpha_t$ is calculated as $\frac{\partial m_i}{\partial \alpha_t} = \frac{b_{i,t}c - a_i}{(c+\alpha_t)^2}$.

Since $c \geq a_i$, depending on the sign of $b_{i,t}$, the derivative of the margin on the $i$th sample $(x_i, y_i)$ is either positive or negative, which is irrelevant to the value of $\alpha_t$. This is also true for the second derivative of the margin. Therefore, the margin on the $i$th example $(x_i, y_i)$ with respect to $\alpha_t$ is either concave when it is monotonically increasing or convex when it is monotonically decreasing. See Figure 2 for a simple illustration.

Consider a greedy coordinate ascent algorithm maximizing the average margin over $n'$ worst training examples, $m_{\text{average n}'}$. Apparently maximizing the minimum margin is a special case by choosing $n' = 1$.

Figure 2 is a simple illustration with six training examples. Our aim is to maximize the average margin over the bottom 3 examples. The interval $[0, d]$ of $\alpha_t$ indicates an interval where the training error is zero. On the point of $d$, the sample $s_4$ alters its margin from positive to negative, which causes the training error to jump from 0 to 1/6. As shown in Figure 2, the margin of six training examples is either monotonically increasing or decreasing.

We have designed an efficient greedy coordinate ascent algorithm that sequentially maximizes the average margin of bottom $n'$ examples, see its details at [25]. We add the weak learner, which has the largest increment of the average margin over bottom $n'$ examples, into the ensembled classifier. This procedure terminates if there is no increment in the average margin over bottom $n'$ examples over all weak learners.

**$\epsilon$-Relaxation**: Unfortunately, there is a fundamental difficulty in the greedy coordinate ascent algorithm that maximizes the average margin of bottom $n'$ samples: It gets stuck at a corner, a coordinatewise maximum solution but not an optimal solution, from which it is impossible to make progress along any coordinate direction. We propose an $\epsilon$-relaxation method [2] to overcome this difficulty. The main idea is to allow a single coordinate to change even if this worsens the margin function. When a coordinate is changed, however, it is set to $\epsilon$ plus or $\epsilon$ minus the value that maximizes the margin function along that coordinate, where $\epsilon$ is a positive number. If $\epsilon$ is small enough, the algorithm can eventually approach a small neighborhood of the optimal solution.

We have also designed a similar greedy coordinate ascent algorithm to directly maximize the bottom $n'$th sample margin.

### 1.1.3 Experimental Results

We evaluate the performance of DirectBoost on 10 UCI data sets and compare with those of AdaBoost [6], LogitBoost [9], LPBoost with column generation [5] and BrownBoost [7]. For all the algorithms in our comparison, we use decision trees with depth of either 1 or 3 as weak learners since for the small datasets, decision stumps (tree depth of 1) is already strong enough. DirectBoost with decision trees is implemented by a greedy top-down recursive partition algorithm to find the tree but differently from AdaBoost and LPBoost, since DirectBoost does not maintain a distribution over training samples. Instead, for each splitting node, DirectBoost simply chooses the attribute to split on by minimizing 0-1 loss or maximizing the predefined margin value. In all the experiments that $\epsilon$-relaxation is used, the value of $\epsilon$ is 0.01.

| Datasets | N | D | depth | AdaBoost | LogitBoost | LPBoost | BrownBoost | DirectBoost$_{\text{avg}}$ | DirectBoost$^{\epsilon}_{\text{avg}}$ | DirectBoost$_{\text{order}}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Tic-tac-toe | 958 | 9 | 3 | 1.47(0.7) | 1.47(1.0) | 2.62(0.8) | 3.66(1.3) | **0.63(0.4)** | 1.15(0.8) | 1.05(0.4) |
| Diabetes | 768 | 8 | 3 | 27.71(1.7) | 27.32(1.3) | 26.01(3.3) | 26.67(2.6) | 25.62(2.5) | 25.49(3.0) | **23.4(3.7)** |
| Australian | 690 | 14 | 3 | 14.2(1.8) | 16.23(2.6) | 14.49(4.4) | 13.77(4.6) | 14.06(3.6) | **13.33(3.0)** | 13.48(2.9) |
| Fourclass | 862 | 2 | 3 | 1.86(1.3) | 2.44(1.6) | 3.02(2.3) | 2.33(1.7) | 2.33(1.0) | 1.86(1.3) | **1.74(1.5)** |
| Ionosphere | 351 | 34 | 3 | 9.71(3.7) | 9.71(3.1) | 8.57(2.7) | 10.86(2.8) | **7.71(3.0)** | 8.29(2.7) | **7.71(4.4)** |
| Splice | 1000 | 61 | 3 | 5.3(1.4) | 5.3(2.6) | 4.8(1.4) | 6.1(1.1) | 4.8(0.7) | **4.0(0.5)** | 6.7(1.6) |
| Cancer-wdbc | 569 | 29 | 1 | 4.25(2.5) | 4.42(1.4) | 3.89(1.5) | 4.25(2.2) | 4.96(3.0) | 4.07(2.0) | **3.72(2.9)** |
| Cancer-wpbc | 198 | 32 | 1 | 27.69(7.6) | 30.26(7.3) | 26.15(10.5) | 28.72(8.4) | 27.69(8.1) | **24.62(7.6)** | 27.18(10.0) |
| Heart | 270 | 13 | 1 | 17.41(7.7) | 18.52(5.1) | 19.26(8.1) | 18.15(7.2) | 18.15(5.1) | **16.67(7.5)** | 18.15(7.6) |
| Adult | 6414 | 14 | 3 | 15.6(0.7) | 15.39(0.8) | 16.2(1.1) | 15.56(0.9) | 16.25(1.7) | **15.28(0.8)** | 15.8(1.1) |

Table 1: Percent test errors of AdaBoost, LogitBoost, soft margin LPBoost with column generation, BrownBoost, and three DirectBoost methods on 10 UCI datasets each with N samples and D variables.

We partition each UCI dataset into five parts with the same number of samples for five-fold cross validation. In each fold, we use three parts for training, one part for validation, and the remaining part for testing. The validation set is used to choose the optimal model for each algorithm: For AdaBoost and LogitBoost, the validation data is used to perform early stopping since there is no nature stopping criteria for these algorithms. We run the algorithms until convergence where the stopping criterion is that the change of loss is less than 1$e$-6, and then choose the ensemble classifier from the round with minimum error on the validation data. For BrownBoost, we select the optimal cutoff parameter by the validation set, and The cutoff parameters for BrownBoost are chosen from {0.0001, 0.001, 0.01, 0.03, 0.05, 0.08, 0.1, 0.14, 0.17, 0.2}. LPBoost maximizes the soft margin subject to linear constraints, its objective is equivalent to the average margin of bottom $n'$ samples [19], thus we set the same candidate parameters $n'/n = \{0.01, 0.05, 0.1, 0.2, 0.5, 0.8\}$ for them. For LPBoost, the termination rule we use is same to the one in [5], and we select the optimal regularization parameter by the validation set. For DirectBoost, the algorithm terminates when there is no increment in the targeted margin value, and we select the model with the optimal $n'$ by the validation set.

We use DirectBoost$_{\text{avg}}$ to denote our method that runs Algorithm 1 first and then maximizes the average of bottom $n'$ margins without $\epsilon$-relaxation, DirectBoost$^{\epsilon}_{\text{avg}}$ to denote our method that runs Algorithm 1 first and then maximizes the average margin of bottom $n'$ samples with $\epsilon$-relaxation, and DirectBoost$_{\text{order}}$ to denote our method that runs Algorithm 1 first and then maximizes the bottom $n'$th margin with $\epsilon$-relaxation. The means and standard deviations of test errors are given in Table 1. Clearly DirectBoost$_{\text{avg}}$, DirectBoost$^{\epsilon}_{\text{avg}}$ and DirectBoost$_{\text{order}}$ outperform other boosting algorithms in general, specially DirectBoost$^{\epsilon}_{\text{avg}}$ is consistently better than AdaBoost, LogitBoost, LPBoost and BrownBoost over all data sets except Cancer-wdbc. Among the family of DirectBoost algorithms, DirectBoost$_{\text{avg}}$ wins on two datasets where it searches the optimal margin solution in the region of zero training error, this means that keeping the training error at zero may lead to good performance in some cases. DirectBoost$_{\text{order}}$ wins on three other datasets, but its results are unstable and sensitive to $n'$. With $\epsilon$-relaxation, DirectBoost$^{\epsilon}_{\text{avg}}$ searches the optimal margin solution in the whole parameter space and gives the best performance on the remaining 5 data sets. It is well known that AdaBoost performs well on the datasets with a small test error such as Tic-tac-toe and Fourclass, it is extremely hard for other boosting algorithms to beat AdaBoost. Nevertheless, DirectBoost is still able to give even better results in this case. For example, on Tic-tac-toe data set, the test error becomes 0.63%, more than half the error rate reduction. Our method would be more valuable for those who value prediction accuracy, which might be the case in areas of medical and genetic research.

Table 2 shows the number of iterations and total run times (in seconds) for AdaBoost, LPBoost and DirectBoost$^{\epsilon}_{\text{avg}}$ at the training stage, where we use the Adult dataset with 10000 training samples. All these three algorithms employ decision trees with a depth of 3 as weak learners. The experiments are conducted on a PC with Core2 Duo 2.6GHz CPU and 2G RAM. Clearly DirectBoost$^{\epsilon}_{\text{avg}}$ takes less time for the entire training stage since it converges much faster. LPBoost converges in less than three hundred rounds, but as a total corrective algorithm, it has a greater computational cost on each round. To handle large scale data sets in practice, similar to AdaBoost, we can use many tricks. For example, we can partition the data into many parts and use distributed algorithms to select the weak learner.

|  | # of iterations | Total running times |
|---|---|---|
| AdaBoost | 117852 | 31168 |
| LPBoost | 286 | 167520 |
| $DirectBoost^{\epsilon}_{avg}$ | 1737 | 606 |

Table 2: Number of iterations and total run times (in seconds) in training stage on Adult dataset with 10000 training samples and the depth of DecisionTrees is 3.

We also have conducted experiments to evaluate the noise robustness. we find that DirectBoost$_{\text{order}}$ has an impressive noise tolerance property.

Please see [25] for more technical detail and experimental results.

## 1.2 Direct Boost for Semi-supervised Classification

Consider semi-supervised binary classification, assume we are given not only a set of $n$ labeled examples, $\mathcal{D}^l = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ but also a set of $m$ unlabeled examples, $\mathcal{D}^u = \{x_{n+1}, \cdots, x_{n+m}\}$. Just as in supervised learning case for boosting, the goal here is that using the combined set of labeled and unlabeled examples $\mathcal{D}^l \cup \mathcal{D}^u$ to construct an ensemble classifier $f \in \mathcal{C}$ that minimizes the following 0-1 loss and has good generalization performance.

$$error(f, \mathcal{D}^l \cup \mathcal{D}^u) = \sum_{i=1}^{n} \mathbf{1}(y_i f(x_i) \le 0) + \gamma \cdot \sum_{i=n+1}^{n+m} \sum_{y \in \mathcal{Y}} p(y|x_i) \mathbf{1}(y f(x_i) \le 0) \tag{5}$$

Here the first term denotes the classification error for labeled data, and the second term represents soft classification error for unlabeled data, $p(y|x_i) = \frac{1}{1+e^{-yf(x_i)}}$ and $\gamma$ is a trade-off parameter that controls the influence of the unlabeled data. The minimum entropy and variance semi-supervised boosting methods [26, 29] optimize the surrogates (log-loss and negative sigmoid function) of (5).

### 1.2.1 Minimizing 0-1 Loss

In semi-supervised case, DirectBoost first runs the algorithm in section 1.1.1 that minimizes the 0-1 loss over labeled data $\mathcal{D}^l$ to construct a good initial enssembled classifier, we then run use an iterative greedy coordinate descent algorithm that directly minimizes (5), and estimate $p(y|x_i)$ through an iterative scheme. That is, given an estimate $\hat{p}_0(y|x_i)$, (5) yields an ensemble classifier $f_1(x_i)$, which leads to a new estimate $\hat{p}_1(y|x_i)$ through Algorithm 1 below. The $\hat{p}_1(y|x_i)$ is expected to be more accurate than $\hat{p}_0(y|x_i)$ for $p(y|x_i)$ because additional information from labeled and unlabeled data has been used in constructing of $f_1(x_i)$ through $\hat{p}_0(y|x_i)$. Consider the $t$th iteration, the ensemble classifier is $f_t(x) = \sum_{k=1}^{t} \alpha_k h_k(x)$, where previous $t-1$ weak classifiers $h_k(x)$ and corresponding weights $\alpha_k, k = 1, \cdots, t-1$ have been selected and determined. We estimate $p(y|x_i)$ by the logit function of the ensemble classifier of the previous step, $\hat{p}(y|x_i) = \frac{1}{1+\exp(yf_{t-1}(x_i))}$. Then the combined 0-1 loss (5) to be a stepwise function of $\alpha_t$.

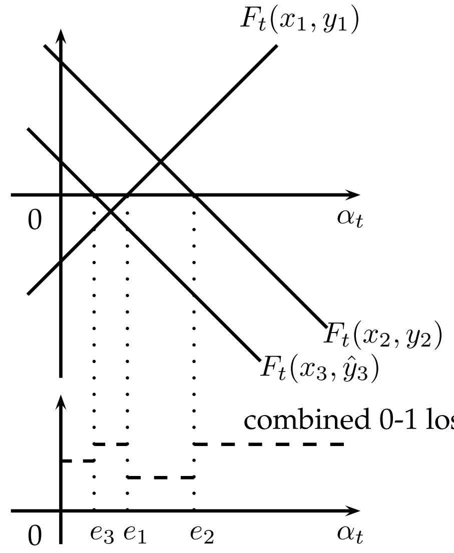

Figure 3: *An example of computing minimum combined 0-1 loss for a weak learner $h_t$ over 2 labeled samples $(x_1, y_1)$ and $(x_2, y_2)$, and 1 unlabeled sample $x_3$.*

Denote $a(x_i) = \sum_{k=1}^{t-1} \alpha_k h_k(x_i)$, then the inference function for sample $x_i$ can be written as,

$$F_t(x_i, y) = y\ h_t(x_i)\alpha_t + ya(x_i) \tag{6}$$

which is a linear function with respect to $\alpha_t$ with slope $yh_t(x_i)$ and intercept $ya(x_i)$. For a labeled example $(x_i, y_i) \in \mathcal{D}^l$, the inference function $F_t(x_i, y_i) > 0$ denotes this example is correctly classified; otherwise, it is misclassified. These two states exchange at point $\alpha_t = -\frac{a(x_i)}{h_t(x_i)}$, which is denoted as the critical point $e_i$, the value of the combined 0-1 loss (5) has $\frac{1}{n}$ difference at $e_i$. For example, in Figure 3, $F_t(x_1, y_1)$ changes its sign from negative to positive at $e_1$, then $(x_1, y_1) \in \mathcal{D}^l$ is correctly classified at $\alpha_t > e_1$ and (5) has $\frac{1}{n}$ reduction on the right side of $e_1$. $F_t(x_2, y_2)$ changes its sign from positive to negative at $e_2$, then $(x_2, y_2) \in \mathcal{D}^l$ becomes misclassified at $\alpha_t > e_2$ and (5) has $\frac{1}{n}$ increment on the right side of $e_2$. To compute the error of unlabeled examples, we use $\hat{y}_i = \text{sign}(a(x_i))$ to denote the pseudo label of an unlabeled example $x_i \in \mathcal{D}^u$. Then, similarly, the sign of $F_t(x_i, \hat{y}_i)$ denotes $x_i \in \mathcal{D}^u$ is "correctly classified" or "misclassified". Again, the critical point for $x_i \in \mathcal{D}^u$ is $\alpha_t = e_i = -\frac{a(x_i)}{h_t(x_i)}$, and the value of the combined 0-1 loss (5) has $\gamma\frac{|\hat{p}(+1|x_i) - \hat{p}(-1|x_i)|}{m}$ difference at $e_i$. In Figure 3, $F_t(x_3, \hat{y}_3)$ changes its sign from positive to negative at $e_3$, and (5) has $\gamma\frac{|\hat{p}(+1|x_3) - \hat{p}(-1|x_3)|}{m}$ increment on the right side of $e_3$. It is obviously that the intercept is always positive for an unlabeled example $x_i \in \mathcal{D}^u$. The critical points $e_i = -\frac{a(x_i)}{h_t(x_i)}$, $i = 1, \cdots, n+m$ divide $\alpha_t$ into (at most) $n+m+1$ intervals, each interval has the value of a combined 0-1 loss (5).

Thus we can design a greedy coordinate descent algorithm that sequentially minimizes (5) [27].

### 1.2.2 Maximizing Margins

Similar to the supervised case, we now describe the algorithm to directly maximize various margins over both labeled and unlabeled examples. The margin of a labeled example $(x_i, y_i) \in \mathcal{D}^l$ w.r.t. an ensemble classifier $f_t(x_i)$ is defined to be

$$\varphi_i^l = \frac{y_i f_t(x_i)}{\sum_{k=1}^t \alpha_k} \tag{7}$$

where $\varphi_i^l$ can be interpreted as a measure of how confident this labeled example is correctly classified. For an unlabeled example $x_i \in \mathcal{D}^u$, we define its margin w.r.t an ensemble classifier $f_t(x)$ as

$$\varphi_i^u = \sum_{y \in \mathcal{Y}} p(y|x_i) \frac{y f_t(x_i)}{\sum_{k=1}^t \alpha_k} = (2p(y=1|x_i) - 1) \frac{f_t(x_i)}{\sum_{k=1}^t \alpha_k} \tag{8}$$

We can sort $\varphi_i^l$ and $\varphi_i^u$ in an increasing order respectively, and consider $n'$ worst labeled examples $n' \leq n$ and $m'$ worst unlabeled examples $m' \leq m$ that have smaller margins, then the combined average margin over those examples is

$$\varphi_{\text{avg}(n',m')} = \frac{1}{n'} \sum_{i \in B_{n'}^l} \varphi_i^l + \gamma \frac{1}{m'} \sum_{i \in B_{m'}^u} \varphi_i^u \tag{9}$$

where $B_{n'}^l$ denotes the set of $n'$ labeled examples having the smallest margins and $B_{m'}^u$ denotes the set of $m'$ unlabeled examples having the smallest margins, and again $\gamma$ is a trade-off parameter that controls the influence of the unlabeled data. $n'$ indicates how much we relax the hard margin on unlabeled examples, and we set $n'$ based on knowledge of the number of noisy examples in $\mathcal{D}^l$ [?]. The higher the noise rate, the larger the $n'$ should be used. $m'$ controls the relaxation of the margin distribution over the unlabeled data. A smaller $m'$ makes the algorithm focus more on the unlabeled examples close to the decision boundary. In this section, we consider (9) as our objective.

For an unlabeled example $x_i \in \mathcal{D}^u$, if we let $p(y|x_i) = \frac{1}{1+e^{-yf_{t-1}(x_i) - \alpha_t y h_t(x_i)}}$, then $\varphi_i^u$ is a complex function of $\alpha_t$. Again use $\hat{p}(y|x_i) = \frac{1}{1+e^{yf_{t-1}(x_i)}}$ instead to estimate the conditional probability by the previous step. Denote $\hat{y}_i = \frac{2}{1+e^{-(f_{t-1}(x_i))}} - 1$, then the estimated margin for $x_i \in \mathcal{D}^u$ is denoted to be

$$\hat{\varphi}_i^u = \hat{y}_i \frac{f_{t-1}(x_i) + \alpha_t h_t(x_i)}{\sum_{k=1}^{t-1} \alpha_k + \alpha_t}, \tag{10}$$

For a given weak hypothesis, we then find $\alpha_t$ maximize (11) instead

$$\hat{\varphi}_{\text{avg}(n',m')} = \frac{1}{n'} \sum_{i \in B_{n'}^l} \varphi_i^l + \gamma \frac{1}{m'} \sum_{i \in B_{m'}^u} \hat{\varphi}_i^u \tag{11}$$

It can be shown that (11) is a quasiconcave function for a given weak hypothesis. This property allows us to design an efficient algorithm that maximizes (11) efficiently.

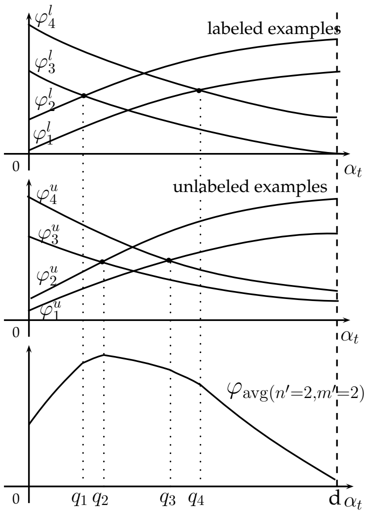

Figure 4: *An example of computing $\alpha_t$ that maximizes $\hat{\varphi}_{\text{avg}(n'=2,m'=2)}$*

The iterative greedy coordinate ascent algorithm that sequentailly and approximately maximizes (9) can be designed, where at each iteration, update $\hat{p}(y|x_i)$, We add the weak hypothesis, which has the largest increment of $\hat{\varphi}_{\text{avg}(n',m')}$, into the ensemble classifier, with the weight that leads to maximum (11). The stopping criterion is that if there is no increment in $\hat{\varphi}_{\text{avg}(n',m')}$ over all weak hypotheses, then the algorithm achieves a stationary point.

Since $\hat{\varphi}_{\text{avg}(n',m')}$ is non-differentiable, a fundamental difficulty in the greedy coordinate ascent algorithm proposed above is that: the algorithm gets stuck at a corner from which it is impossible to make progress along any coordinate direction. To overcome this difficulty, we employ an $\epsilon$-relaxation method. The main idea is to allow a single coordinate to change even if this worsens the margin function. When a coordinate is changed, however, it is set to $\epsilon$ plus or $\epsilon$ minus the value that maximizes the margin function along that coordinate, where $\epsilon$ is a positive number.

### 1.2.3 Experimental Results

We compare the performance of SSDirectBoost with those of AdaBoost, DirectBoost, ASSEMBLE [1], and EntropyBoost [29] on UCI datasets. Since we concentrate on the case that the labeled data are limited while the unlabeled data are adequate in the training process, we randomly select a small portion of data as the labeled training examples, the remaining examples are used as unlabeled training data, validation data and testing data. The dimension of data and the number of separate labeled (L), unlabeled (U), validation (V), and test (T) examples for each dataset are given in Table 3. We use the validation data to choose the optimal model for each algorithm. For AdaBoost, the validation data is used to perform early stopping. We run AdaBoost until convergence where the stopping criterion is that the change of loss is less than 1*e*-6, and then choose the ensemble classifier from the round with minimum error on the validation data. The early stopping technique is applied on ASSEMBLE, and EntropyBoost since there is no nature stopping criteria. Moreover, for ASSEMBLE and EntropyBoost, the tradeoff parameters that control the influence of unlabeled data are chosen by the validation set on the values $\{1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$. For DirectBoost, the parameter $n'$ is chosen by the validation data on the values $\{1, n/5, n/3, n/2\}$. For SSDirectBoost, the parameter $n'$ is chosen on the values $\{1, n/5, n/3, n/2\}$ and $m'$ is chosen on the values $\{1, m/3, m/2\}$ by the validation set, and $\gamma$ is set to 0.1. The stopping criterion of SSDirectBoost is defined as line 12 in Algorithm 2, SSDirectBoost terminates at the margin maximization solution, thus we need not apply early stopping on validation data.

| Data | Dim. | No. of examples | | | | Depth | AdaBoost | DirectBoost | ASSEMBLE | EntropyBoost | SSDirectBoost |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | L | U | V | T | | | | | | |
| Mushroom | 22 | 20 | 1000 | 50 | 7054 | 1 | 8.81(1.9) | 5.38(1.8) | 5.05(0.7) | 5.1(1.6) | **2.2(0.5)** |
| Adult | 14 | 50 | 1000 | 50 | 47742 | 1 | 20.33(2.0) | 20.14(1.9) | **19.77(1.8)** | 20.53(2.2) | 19.9(2.0) |
| Australian | 14 | 50 | 300 | 40 | 300 | 1 | 15.67(1.1) | 15.0(0.5) | 14.73(1.1) | 14.73(0.9) | **13.67(0.9)** |
| Liver | 6 | 30 | 200 | 115 | 200 | 1 | 41.5(4.3) | 41.1(5.9) | 36.9(5.7) | 37.2(5.2) | **36.3(5.3)** |
| Sonar | 60 | 20 | 100 | 88 | 100 | 1 | 33.8(3.8) | 33.6(4.7) | 31.8(5.1) | 35.4(4.4) | **28.0(2.5)** |
| Kr-vs-Kp | 36 | 50 | 1000 | 50 | 2096 | 1 | 10.46(2.2) | 8.66(2.4) | 8.2(2.2) | 8.5(2.1) | **7.65(2.0)** |
| Cod-Rna | 7 | 50 | 1000 | 50 | 58435 | 3 | 17.33(2.4) | 16.71(2.6) | 18.87(2.7) | 19.6(3.0) | **14.44(1.8)** |
| Splice | 61 | 100 | 400 | 100 | 400 | 3 | 13.32(1.0) | 12.96(2.2) | 14.12(2.0) | 14.04(1.1) | **10.72(2.1)** |
| Magic | 100 | 100 | 1000 | 100 | 17820 | 3 | 20.6(2.0) | 19.72(1.3) | 19.87(1.3) | 20.8(1.2) | **19.51(1.1)** |
| Spambase | 57 | 100 | 1000 | 100 | 3401 | 3 | 10.2(1.5) | 11.0(1.0) | 10.55(0.8) | 10.45(1.0) | **8.96(0.9)** |

Table 3: Mean error rates (in %) and standard deviation of each boosting method on UCI datasets when decision trees (with depth of 1 or 3) are used as weak learners.

Table 3 shows the results of different boosting methods when decision trees (with depth of 1 or 3) are used as weak learners. As we expected, semi-supervised boosting algorithms outperform the supervised methods, the results indicate that the unlabeled data does help to improve generalization performance. Furthermore, the proposed SSDirectBoost overcomes the gradient based semi-supervised boosting methods in general by taking advantage of maximizing the margin objective function directly on $\mathcal{D}^l \cup \mathcal{D}^u$. When decision trees with depth of three are used, we noticed that ASSEMBLE and EntropyBoost sometimes perform worse than supervised boosting algorithms, but our proposed SSDirectBoost consistently gives significantly better results in this case.

Please see [27] for more technical detail and experimental results.

## 1.3 Direct Boost for Multi-class Classification

In multi-class classification, we want to predict the labels of examples lying in some set $\mathcal{X}$. We are provided a training set of labeled examples $\mathcal{D} = (x_1, y_1), \cdots, (x_n, y_n)\}$, where each example $x_i \in \mathcal{X}$ has a unique $y_i$ label in the set $\{1, \cdots, K\}$. Again denote $\mathcal{H} = \{h_1, ..., h_l\}$ as the set of all possible weak classifiers that can be produced by the weak learning algorithm, where a weak classifier $h_j \in \mathcal{H}$ is a mapping from an instance space $\mathcal{X}$ to $\mathcal{Y} = \{1, \cdots, K\}$. Boosting combines weak classifiers to form a highly accurate combined classifier for multiclass classification by making a prediction according to the weighted plurality vote of the classifiers:

$$\hat{y} \quad = \quad \arg\max_{y \in \{1, \cdots, K\}} f(x, y), \tag{12}$$

where $f(x, y) = \sum_{h \in \mathcal{H}} \alpha_h \mathbf{1}(h(x) = y), \alpha_h \in \mathcal{R}$. The empirical error for a multi-class classification problem is given by (2). Our goal is to find $\underline{f} = (f(x, y), y \in \mathcal{Y})$ that attains a small empirical error on $\mathcal{D}$ and also generalizes well. In the following, we describe novel methods that directly minimize (2) and maximize various margins.

### 1.3.1 Minimizing 0-1 Loss

Similar to the binary classification, we use a greedy coordinate descent algorithm to directly minimize the empirical error (2) and construct an ensembled classifier. Consider the $t$th iteration, the ensemble classifier is $f_t(x, y) = \sum_{k=1}^{t} \alpha_k \mathbf{1}(h_k(x) = y)$, $\forall y \in \mathcal{Y}$, where previous $t-1$ weak classifiers $h_k(x)$ and corresponding weights $\alpha_k$, $k = 1, \cdots, t-1$ have been selected and determined. Let $a(x_i, y) = \sum_{k=1}^{t-1} \alpha_k \mathbf{1}(h_k(x) = y)$. We define the inference functions for example $x_i$ as

$$F_t(x_i, y) = f_t(x_i, y) = a(x_i, y) + \alpha_t \mathbf{1}(h_t(x) = y), \tag{13}$$

which is a linear function of $\alpha_t$ with intercept $a(x_i, y)$ and slope $\mathbf{1}(h_k(x) = y)$. Obviously, the inference function is either a line with slope 1 or a horizontal line. The inference functions are used to compute the empirical error (2). More specifically, given a weak learner $h_t \in \mathcal{H}$, for each example pair $(x_i, y_i)$, there are 3 scenarios to compute the empirical error, see Figure 5. Scenario 1 is the case that $h_t(x_i) = y_i$. $F_t(x_i, y_i)$ is a line with slope 1, and assume that $l = \arg\max_{y \in \mathcal{Y}, y \neq y_i} a(x_i, y)$, then $F_t(x_i, l)$ is a line with slope 0. The intersection of $F_t(x_i, y_i)$ and $F_t(x_i, l)$ is at $\alpha_t = a(x_i, l) - a(x_i, y_i)$. Thus when $\alpha_t$ is set on the left side of the intersection point, there is an error for example $x_i$ and otherwise there is no error. Scenario 2 is the case that $h_t(x_i) = y$, $y \neq y_i$, and $a(x_i, y_i) > a(x_i, y)$ $\forall y \in \mathcal{Y}$, $y \neq y_i$. Then $F_t(x_i, y)$ is a line with slope 1, and $F_t(x_i, y_i)$ is a line with slope 0. The intersection point of $F_t(x_i, y)$ and $F_t(x_i, y_i)$ is at $\alpha_t = a(x_i, y) - a(x_i, y_i)$. Thus when $\alpha_t$ is set on the right side of the intersection point, there is an error for example $x_i$ and otherwise there is no error. Scenario 3 is the case that $h_t(x_i) = y$, and $y \neq y_i$, and $\exists l \in \mathcal{Y}$, $l \neq y_i$ such that $a(x_i, l) > a(x_i, y_i)$, in this case there is always an error no matter what value $\alpha_t$ is.

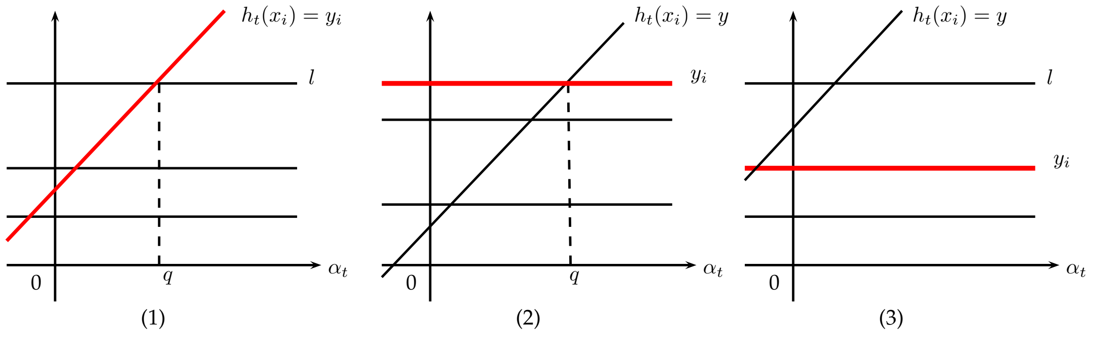

Figure 5: *Three scenarios to compute the empirical error of a weak learner $h_t$ over an example pair $(x_i, y_i)$, where $l$ denotes the incorrect label with highest score, and $p$ denotes the intersection point that results empirical error change. The red, bold line for each scenario represents the inference function of example $x_i$ and its true label $y_i$.*

### 1.3.2 Maximizing Margins

Similar to binary classification, we now describe the algorithm that directly maximizes various margins for multi-class classification. Define the margin of a labeled example $(x_i, y_i)$ with respect to an ensembled multi-class classifier $f_t(x, y) = \sum_{k=1}^{t} \alpha_k \mathbf{1}(h_k(x_i) = y), \forall y \in \mathcal{Y}$ to be

$$m_i = \frac{f_t(x, y_i)}{\underline{\alpha}_{l_1}} - \max_{y \in \mathcal{Y}, y \neq y_1} \frac{f_t(x, y)}{\underline{\alpha}_{l_1}} \tag{14}$$

We can sort $m_i$ in an increasing order, and consider $n'$ worst training examples $n' \leq n$ that have smaller margins, then define the average margin over those $n'$ labeled examples by $g_{avg\ n'}$. Formally,

$$g_{avg\ n'} = \frac{1}{n'} \sum_{i \in B_{n'}} m_i \tag{15}$$

where $B_{n'}$ denotes the set of $n'$ labeled examples having the smallest margins.

We have designed an algorithm to maximize (15). Given a weak learner $h_t \in \mathcal{H}$ at $t$th iteration, let $c = \sum_{k=1}^{t-1} |\alpha_k|$, then the margin on the example $(x_i, y_i)$ can be rewritten as,

$$m_i = \frac{a(x_i, y_i) + \alpha_t \mathbf{1}(h_t(x_i) = y_i)}{c + |\alpha_t|} - \max_{y \in \mathcal{Y}, y \neq y_i} \frac{a(x_i, y) + \alpha_t \mathbf{1}(h_t(x_i) = y)}{c + |\alpha_t|} \tag{16}$$

Consider the case that $\alpha_t \geq 0$. For each example pair $(x_i, y_i)$, there are three scenarios of (16) to consider, as shown in Figure 6. Scenario 1 is the case that $h_t(x_i) = y_i$, and assume that $l = \arg\max_{y \in \mathcal{Y}, y \neq y_i} a(x_i, y)$, then $m_i = \frac{a(x_i,y_i) - a(x_i,l) + \alpha_t}{c + \alpha_t}$. This corresponds to the curve which is monotonically increasing in Figure 6. Scenario 2 is the case that $h_t(x_i) = l$, $y \neq y_i$, and $a(x_i, l) > a(x_i, y)$, $\forall y \in \mathcal{Y}$, $y \neq y_i$, then $m_i = \frac{a(x_i,y_i) - a(x_i,l) - \alpha_t}{c + \alpha_t}$. This corresponds to the curve which is monotonically decreasing in Figure 6. Scenario 3 is the case that $h_t(x_i) = y$, and $y \neq y_i$, and $\exists l \in \mathcal{Y}$, $l \neq y_i$ such that $a(x_i, l) > a(x_i, y)$, in this case the margin curve of $m_i$ has two pieces. When $\alpha_t < a(x_i, y) - a(x_i, l)$, $m_i = \frac{a(x_i,y_i) - a(x_i,l)}{c + \alpha_t}$ and when $\alpha_t > a(x_i, y) - a(x_i, l)$, $m_i = \frac{a(x_i,y_i) - a(x_i,y) - \alpha_t}{c + \alpha_t}$. The scenarios for the case that $\alpha_t < 0$ can be similarly identified.

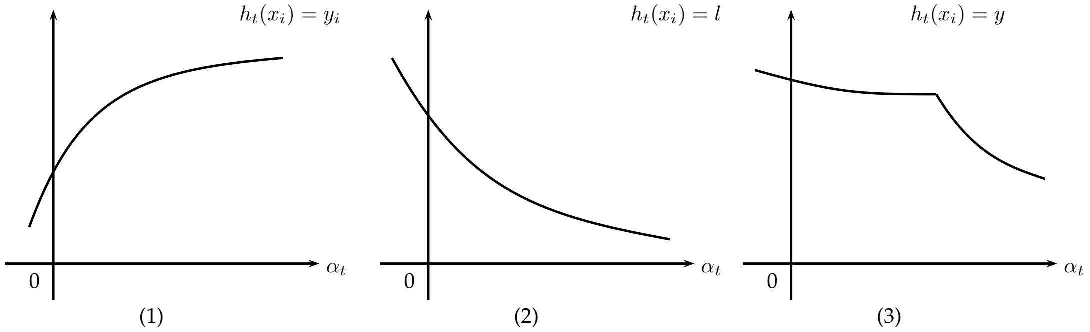

Figure 6: *Three scenarios of margin curve of a weak learner $h_t$ over an example pair $(x_i, y_i)$.*

Thus in the margin maximization phase, the key step is to find the value of $\alpha_t$ within an interval $[0, d]$ that maximize (15) for a given $h_t$. Finding the exact solution is computationally difficult since the examples in Scenario 3 can either intersect with the examples in Scenario 1 or intersect with the examples in Scenario 2. Fortunately, we can prove that (15) is quasi-concave, which allows us to design a line search algorithm that maximizes (15) efficiently by checking the derivative of (15). We have designed a greedy coordinate ascent algorithm that sequentially maximizes the average margin of bottom $n'$ examples, it terminates if there is no increment in the average margin over bottom $n'$ examples over $h_t$. Again since (15) is non-differentiable at turning points, the coordinate ascent algorithm may get stuck at a corner from which it is impossible to make progress along any coordinate direction. To overcome this difficulty, we use an $\epsilon$-relaxation method, which allows a single coordinate to change even if this worsens the objective value. When a coordinate is changed, it is set to $\epsilon$ plus (or $\epsilon$ minus) the value that maximizes the objective function along that coordinate, where $\epsilon$ is a small positive number. If $\epsilon$ is small enough, the algorithm can eventually approach a small neighborhood of the optimal solution.

### 1.3.3 Experimental Results

To evaluate the performance of the MCDB algorithm, we conduct experiments with 12 UCI datasets. For comparison, we also report the results of AdaBoost.M1 [6], AdaBoost.MH [16], SAMME [30], and GD-MCBoost [14]. All these algorithms use multi-class base classifiers except AdaBoost.MH, which reduces the multi-class problem to a set of binary classification problems. The classification error is estimated either by a test error or five-fold cross-validation. The datasets come with pre-specified training and testing sets are evaluated by test error, where $n'$ is set to $\frac{n}{4}$ for MCDB and the number of rounds is set to maximum of 5000 for each method. For datasets which are evaluated by cross-validation, we partition them into five parts evenly for 5-fold. In each fold, we use three parts for training, one part for validation, and the remaining part for testing. We use the validation data to choose the optimal model for each algorithm. For AdaBoost.M1, AdaBoost.MH, SAMME, and GD-MCBoost, the validation data is used to perform early stopping. We run these algorithms with a maximum of 5000 iterations, and then choose the ensemble classifier from the round with minimum error on the validation data. For MCDB, the parameter $n'$ is chosen on the values $\{1, \frac{n}{10}, \frac{n}{5}, \frac{n}{4}, \frac{n}{3}, \frac{n}{2}, \frac{2n}{3}\}$ by the validation set. The stopping criterion of MCDB is defined as line 8 in Algorithm 3 where MCDB terminates at the margin maximization solution, thus we need not to apply early stopping. In all the experiments, the value of $\epsilon$ is set to be 0.01 and the value of $th$ is set to be $1e$-5.

An overview of these datasets is shown in Table 4. In the # Examples column, the number of training/test examples are listed for datasets coming with pre-specified training and testing sets, and the entire number of examples are given for the rest datasets. The original Poker dataset has 25,010 training examples and 1,000,000 examples for testing. Since the test data is very large, we randomly divide it equally into two parts, and add them to training and testing sets respectively, thus its training size becomes 525,010 and test size becomes 500,000.

| Data | # Examples | K | # Variables | Error Estimation |
|---|---|---|---|---|
| Abalone | 4177 | 28 | 8 | 5-CV |
| Car | 1728 | 4 | 6 | 5-CV |
| Krkopt | 28056 | 18 | 6 | 5-CV |
| Letter | 20000 | 26 | 16 | 5-CV |
| Nursery | 12960 | 5 | 8 | 5-CV |
| Poker525k | 525010/500000 | 10 | 11 | test error |
| Segmentation | 210/2100 | 7 | 19 | test error |
| Waveform | 5000 | 3 | 21 | 5-CV |
| Yeast | 1484 | 10 | 8 | 5-CV |
| Glass | 214 | 6 | 10 | 5-CV |
| Wine | 178 | 3 | 13 | 5-CV |
| Vowel | 990 | 11 | 10 | 5-CV |

Table 4: Description of datasets

First, we restrict the base classifiers to smaller trees to test the performance of each algorithm when the base classifiers are very weak. We exclude the results of AdaBoost.MH as all the rest algorithms use multi-class base classifiers, and we want to compare the performance of each algorithm with the same hypothesis space $\mathcal{H}$. Table 5 shows the results of different methods when multi-class decision trees with a depth of 3 are used as weak learners. With smaller trees, MCDB gives the best results on all datasets indicating that MCDB only requires very weak base classifiers even if there is no exact weak learner condition for MCDB. GD-MCBoost achieves the second best accuracy, and this algorithm also requires weaker base classifiers since it is able to boost any type of weak learners with non-zero directional derivatives [14]. We do not report its results on Poker525k dataset since its one iteration takes more than 12 hours to run by authors' matlab code. For SAMME, the weak learner conditions can be satisfied easily, but it couldn't drive down the training error when the base classifier is very weak, and its performance is much worse. AdaBoost.M1 gives the worst results, and it is not able to boost the base classifiers for 4 of 12 datasets, as shown in Table 5.

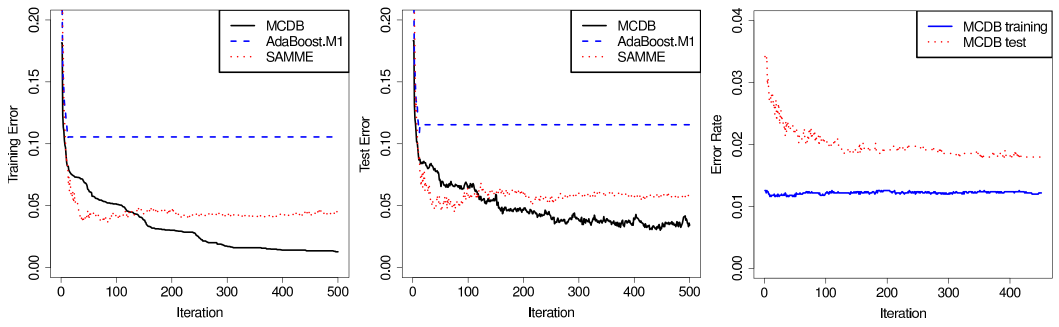

Figure 7: *Learning curves on training data, test data by Algorithm 1 and 3 respectively.*

| Data | AdaBoost.M1 | SAMME | GD-MCBoost | MCDB |
|---|---|---|---|---|
| Abalone | - | 74.20(1.8) | 74.62(1.5) | **74.03(2.0)** |
| Car | 10.96(2.5) | 4.75(1.0) | 3.60(1.1) | **2.78(0.8)** |
| Krkopt | - | 64.33(0.9) | 26.55(0.4) | **22.76(0.7)** |
| Letter | - | 24.94(0.9) | 5.40(1.3) | **4.89(0.3)** |
| Nursery | 9.70(1.5) | 3.26(0.7) | 0.2(0.0) | **0.02(0.0)** |
| Poker525k | 49.16 | 69.09 | - | **30.09** |
| Segmentation | 8.29 | 6.43 | 6.0 | **5.1** |
| Waveform | 17.8(1.2) | 16.96(1.2) | 16.2(1.1) | **14.38(1.1)** |
| Yeast | 43.65(2.6) | 44.73(4.5) | 43.6(3.5) | **42.43(2.8)** |
| glass | 29.52(10.7) | 31.9(8.0) | 27.0(7.4) | **26.19(10.8)** |
| wine | 8.57(4.9) | 7.43(4.8) | 7.54(5.3) | **3.43(4.7)** |
| vowel | - | 19.19(2.6) | 9.2(2.6) | **5.66(1.9)** |

Table 5: Test error (and standard deviation) of multi-class boosting methods on UCI datasets, using decision trees with a depth of 3.

With the same hypothesis space $\mathcal{H}$ (trees with a depth of 3), Algorithm 1 usually achieves a lower training classification error rate. The left panel of Figure 7 shows a typical training error curve on car dataset, and the middle panel shows the corresponding test error curve. Once Algorithm 1 terminates at a coordinatewise local minimum, Algorithm 3 can still drive down the test error even when the training error does not decrease, as shown in the right panel of Figure 7.

We next investigate how these algorithms perform with more powerful base classifiers. We tried all tree depths in the candidate set {3,5,8,12} for each dataset. This time we compare the algorithms not restricted in the same hypothesis space $\mathcal{H}$, so we also add AdaBoost.MH in the comparison. As shown in Table 6, among all the methods, MCDB gives the most accurate results in 9 of the 12 datasets, and its results are close to the best results produced by other methods for the remaining 3 datasets.

Please see [28] for more technical detail and experimental results.

## 1.4 Direct Optimization for Ranking

First we describe DirectRank that learns a linear ranking function by directly optimizing any ranking measures. Suppose that a set of training queries $Q_s = \{q_1, q_2, \cdots q_n\}$ is given, and a set of documents $\mathbf{d}_i = \{d_{i1}, d_{i2}, \cdots, d_{i,m(q_i)}\}$ is retrieved for each query $q_i$. Let $m(q_i)$ denote the size of the set of retrieved documents, which varies for different queries. Every document $d_{ij}$ is associated with a manually-labeled judgment $y_{ij} \in \{r_1, r_2, \cdots, r_l\}$, that denotes the relevance of a document to the query. We define the order $r_l \succ r_{l-1} \succ \cdots \succ r_2 \succ r_1$, where $\succ$ means the preference relationship. A $L$-dimensional feature

| Data | AdaBoost.M1 | AdaBoost.MH | SAMME | GD-MCBoost | MCDB |
|---|---|---|---|---|---|
| Abalone | 76.41(1.4) | 75.33(1.2) | 73.70(1.7) | 74.62(1.5) | **73.44(1.8)** |
| Car | 3.36(0.8) | 2.84(0.6) | 3.65(0.9) | 2.8(0.8) | **2.67(0.8)** |
| Krkopt | 14.3(0.3) | 11.68(0.3) | 12.71(0.2) | 12.20(0.3) | **11.04(0.2)** |
| Letter | 3.48(0.3) | **3.1(0.1)** | 4.88(0.3) | 3.37(0.2) | **3.1(0.2)** |
| Nursery | 0.12(0.1) | 0.03(0.0) | 0.16(0.1) | **0.0(0.0)** | **0.0(0.0)** |
| Poker525k | 30.19 | **2.01** | 18.74 | - | 2.77 |
| Segmentation | 4.86 | 6.14 | 5.1 | 6.0 | **4.52** |
| Waveform | 15.2(1.4) | 14.56(1.4) | 15.08(1.0) | 15.2(0.8) | **14.26(1.1)** |
| Yeast | 41.69(1.8) | 41.82(2.1) | 41.22(3.1) | 43.2(3.6) | **40.23(2.5)** |
| glass | 27.14(9.3) | 29.52(9.2) | 24.76(8.7) | **24.0(6.8)** | 24.76(9.9) |
| wine | 8.57(4.9) | 9.16(5.3) | 7.43(4.8) | 7.54(5.3) | **3.43(4.7)** |
| vowel | 5.96(2.9) | 7.68(1.8) | 6.25(2.3) | **5.6(3.0)** | 5.66(1.9) |

Table 6: Test error (and standard deviation) of multi-class boosting methods on UCI datasets, using decision trees with a maximum depth of 12.

vector is created for each query-document pair $(q_i, d_{ij}), i = 1, \cdots, n, j = 1, \cdots, m(q_i)$ and is denoted as $\mathbf{g}(d_{ij}|q_i) = (g_1(d_{ij}|q_i), \cdots, g_L(d_{ij}|q_i))$.

The objective of ranking is to construct a ranking function $f$ such that for each query the retrieved documents can be assigned ranking scores using the function and then be ranked according to the scores. The learning process turns out to be that of optimizing the ranking measure which represents the agreement between the permutation by relevance judgments and the ranking yielded by a ranking function. We use the linear ranking function,

$$f(\mathbf{g}(d_{ij}|q_i)) = \underline{\alpha} \cdot \mathbf{g}(d_{ij}|q_i)) \tag{17}$$

where the weight vector $\underline{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_L)$ is the model parameter. *Assume the ranking measure is NDCG.* Since NDCG is non-convex, non-differentiable and discontinuous with respect to $\underline{\alpha}$, thus we cannot use standard optimization algorithms such as gradient ascent to optimize directly.

DirectRank is an iterative coordinate ascent method to directly optimize NDCG. For each iteration, there will be only one coordinate parameter updated, denoted as $\alpha_k$, while others keep unchanged. The rationale of this idea is that the ranking function is written as a one-dimensional linear function,

$$f(\mathbf{g}(d_{ij}|q_i)) = \alpha_k \cdot g_k(d_{ij}|q_i) + \sum_{l \neq k}^{L} \alpha_l g_l(d_{ij}|q_i)$$

Since $g_k(d_{ij}|q_i)$ is constant with respect to $\alpha_k$, and so is the second term, we can re-write these two quantities as $b_{ij}$ and $a_{ij}$, and convert the equation above to,

$$f(\mathbf{g}(d_{ij}|q_i)) = b_{ij} \cdot \alpha_k + a_{ij} \tag{18}$$

Note that for each document $d_{i,j}$ retrieved by each query $q_i$, there is a linear function of $\alpha_k$. Given an input of $\alpha_k$, each document will get an output

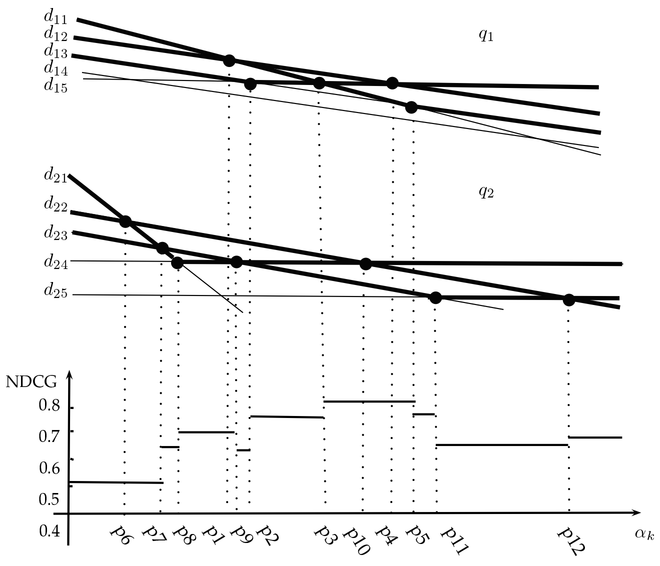

Figure 8: *The top-$\tau$ ($\tau$=3) candidates for each of the two queries are marked as bold. Between each two boundaries, the NDCG value is shown. We can see the intervals between $p_3$ and $p_5$ achieve the best NDCG.*

score from this linear function. The order of such scores actually reflects the order of the documents which further determines the NDCG value.

This is illustrated in Figure 8, where each of the lines represents a scoring function for a document. At any point of $\alpha_k$, the rank of the linear function output scores is equivalent to the rank of the documents. Note that a little change of $\alpha_k$ cannot lead to a jump of NDCG value, unless it changes the order of the top-$\tau$ documents. Such change in order happens only at the point where two lines intersect. We denote the set of such points as *jumping points*. In Figure 8, we draw ten lines corresponding to ten documents, which belong to two queries of the top-3 ranked documents. $(p_1, p_2, \cdots p_{12})$ are jumping points.

Theoretically, we can search all the intersections to acquire all possible snapshots of ranked documents. Because any two of the non-parallel lines will form an intersection, the total number of intersections then is $m(q_i)^2$. When $m(q_i)$ is large, this effort is not only time-consuming but also completely unnecessary, because in real-world applications we are merely interested in the rank of top-$\tau$ candidates, the NDCG metric is always truncated to a certain level $\tau$, and usually $\tau \leq 10$. As a result, the jumping point size between top-$\tau$ candidates is quite limited, and does not increase linearly as the document size increases. Therefore, we can efficiently find all the jumping points on one coordinate and find the interval which achieves the optimal NDCG value. Then we determine the optimal parameter value by maximizing the likelihood of top-$\tau$ ranked documents for a ranking by human judgment within these intervals.

The decision tree representation for the features of the query-document pair is capable of capturing more complex relations between original features, and has been used by LambdaMart [22] to significantly improve the ranking performance over LambdaRank. Thus we integrate regression trees into DirectRank effectively and conveniently by following a *stage-wise* strategy. We use MART trees [10] as our weak learners, and in order to enhance the stability, we restrict the new weight $\alpha_k$ in range $[a, b]$, where we empirically set the hyper-parameters between $[0.1, 0.5]$ in our experiments. If the output is beyond the range, we just take the border values.

### 1.4.1 Experimental Results

We have applied DirectRank to two large datasets, Yahoo Challenge Track 1 data and Microsoft 30K web data. We achieved the best results. For example, for the Yahoo Challenge Track 1 dataset, since we use a linear function in DirectRank, to have a fair comparison, we compare it with LambdaRank [3], whose ranking function is also linear and has the best reported result. Table 7 shows that DirectRank (DR) outperforms LambdaRank (LR). We also compare DirectRank with other baselines, such as SmoothGrad (SG) [11], AdaRank (AR) [24], ad hoc coordinate ascent (CA) [13], RankBoost (RB) [8] and ListNet (LN) [4]. Table 8 shows the running times for these methods. Given a randomly generated starting point, DirectRank converges after approximately 20 rounds and takes a total time of 2.3 hours. SmoothGrad is the fastest, however, it does not perform as well as DirectRank.

| | DR | LR | SG | AR | CA | RB | LN |
|---|---|---|---|---|---|---|---|
| TRAIN | **0.762** | – | 0.741 | 0.728 | 0.750 | 0.734 | 0.709 |
| VALID | **0.757** | – | 0.738 | 0.723 | 0.744 | 0.730 | 0.700 |
| TEST | **0.760** | 0.757 | 0.739 | 0.729 | 0.745 | 0.732 | 0.705 |

Table 7: NDCG@10 on Yahoo Challenge Track 1 dataset.

| | DR | SG | AR | CA | RB | LN |
|---|---|---|---|---|---|---|
| HOURS | 2.3 | 0.3 | 11.8 | 45.3 | 24.5 | 23.8 |

Table 8: Running time on Yahoo Challenge Track 1 dataset.

As tree-based models generally outperform linear models, we compare our system with two state-of-the-art systems, MART [10] and LambdaMART on two large datasets. The maximum number of trees is set to 1000. In Yahoo data, the number of leaf nodes is set to 10, and more leaves do not contribute to the final performance significantly with respect to the official measure NDCG@10. On Microsoft 30K web data, we adjust the number of leaf nodes as 10, 30 and 50.

| | @1 | @10 |
|---|---|---|
| MT | .7084 | .7768 |
| LM | .7167 | .7791 |
| DirectRank | **.7199** | **.7810** |

Table 9: NDCG scores of tree models on Yahoo Challenge, including MART (MT), LambdaMART (LM).

| | @1 | | | @10 | | |
|---|---|---|---|---|---|---|
| | MT | LM | DR | MT | LM | DR |
| 10 | .4582 | .4602 | **.4894** | .4887 | .4943 | **.4985** |
| 30 | .4823 | .4830 | **.4917** | .4994 | .4997 | **.5055** |
| 50 | .4744 | .4883 | **.4911** | .5022 | .5006 | **.5061** |

Table 10: NDCG score of tree models on Microsoft 30K web data with varying number of leaf nodes, including MART(MT), LambdaMART (LM), DirectRank (DR).

DirectRank shows significant superiority to NDCG@1 over Microsoft 30K web data, especially when the number of leaf nodes is quite small, and in other cases (Table 9 and 10) DirectRank still performs slightly better. We find the average number of documents per query is greatly different in the two datasets, about 23 in Yahoo dataset and 72 in Microsoft data. Since MART treats all documents equally, more documents may in some sense have a negative influence on the objective NDCG@1; thus, it would be more likely to acquire improvement by adopting an accurate objective. When the number of leaf nodes increases, the two baselines improve significantly in NDCG@1, while our DirectRank is more stable and effective in performance. Moreover, even for a small number of leaf nodes, DirectRank works very well. Finally, MART in [21] gets a higher performance by using a complete binary tree with different depths, and all tree-based algorithms here are implemented in a fair manner by restricting the maximum number of leaf nodes.

Please see [19] for more technical detail and experimental results.

## References

[1] K. Bennett, A. Demiriz and R. Maclin. Exploiting unlabeled data in ensemble methods. *International Conference on Knowledge Discovery and Data Mining* (KDD), 2002.

[2] D. Bertsekas. *Network Optimization: Continuous and Discrete Models.* Athena Scientific, 1998.

[3] C. Burges, R. Ragno and Q. Le. Learning to Rank with Nonsmooth Cost Functions. *Advances in Neural Information Processing* (NIPS), 19:193-200, 2007

[4] Z. Cao, T. Qin, T. Liu, M. Tsai and H. Li. Learning to Rank: From Pairwise to Listwise Approach. *ICML*, 129-136, 2007

[5] A. Demiriz, K. Bennett and J. Shawe-Taylor. Linear programming boosting via column generation, *Machine Learning*, 46:225–254, 2002.

[6] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[7] Y. Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43(3):293–318, 2001.

[8] Y. Freund, R. Iyer, R. Schapire and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *JMLR*, 4:933-969, 2003

[9] J. Friedman, T. Hastie and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.

[10] J. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189-1232, 2001

[11] Q. Le and A. Smola. Direct Optimization of Ranking Measures. *NICTA Tech report*, 2007

[12] C. Lu, W. Wang, M. Nagarajan, S. Wang and A. Sheth. Extracting diverse sentiment expressions with target-dependent polarity from Twitter. *The Sixth International AAAI Conference on Weblogs and Social Media* (ICWSM), 50-57, 2012.

[13] D. Metzler and W. Croft. Linear Feature-Based Models for Information Retrieval. *Information Retrieval*, 10(3):257-274, 2007

[14] M. Saberian and N. Vasconcelos. Multiclass boosting: Theory and algorithms. *Neural Information Processing Systems* (NIPS), 2011.

[15] R. Schapire, Y. Freund, P. Bartlett and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[16] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[19] S. Shalev-Shwartz and Y. Singer. On the equivalence of weak learnability and linear separability: new relaxations and efficient boosting algorithms. *Machine Learning*, 80(2-3): 141-163, 2010.

[17] M. Tan, W. Li, L. Zheng and S. Wang. A large scale distributed syntactic, semantic and lexical language model for machine translation. *The 49th Annual Meeting of the Association for Computational Linguistics and Human Language Technologies* (ACL/HLT), 201-210, 2011.

[18] M. Tan, W. Li, L. Zheng and S. Wang. A scalable distributed syntactic, semantic and lexical language model. *Computational Linguistics*, 3:631-671, 2012.

[19] M. Tan, T. Xia, L. Guo and S. Wang. Direct optimization of ranking measures for learning to rank models. *The 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (KDD), 2013.

[20] M. Tan, T. Xia, S. Wang and B. Zhou. A corpus level MIRA tuning strategy for machine translation. *Conference on Empirical Methods in Natural Language Processing* (EMNLP), 2013.

[21] S. Tyree, K. Weinberger, K. Agrawal and J. Paykin. Parallel boosted regression trees for web search ranking. *The 20th International Conference on World Wide Web Pages*, (WWW), 387-396, 2011

[22] Q. Wu, C. Burges, K. Svore, and J. Gao. Adapting Boosting for Information Retrieval Measures. *Information Retrieval*, 13(3):254-270, 2010

[23] T. Xia, M. Tan, S. Zhai and S. Wang. Improving alignment of system combination by using multi-objective optimization. *Conference on Empirical Methods in Natural Language Processing* (EMNLP), 2013.

[24] J. Xu and H. Li. AdaRank: A Boosting Algorithm for Information Retrieval. *SIGIR*, 391-398, 2007

[25] S. Zhai, T. Xia, M. Tan and S. Wang. Direct 0-1 loss minimization and margin maximization with boosting. *Advances in Neural Information Processing* (NIPS), 2013.

[26] S. Zhai, T. Xia, M. Tan and S. Wang. A robust semi-supervised boosting method using linear programming. *IEEE GlobalSIP Symposium on Optimization in Machine Learning and Signal Processing*, 2013.

[27] S. Zhai, M. Tan, T. Xia and S. Wang. Semi-supervised boosting with direct 0-1 loss minimization and margin maximization. Technical Report, 2013.

[28] S. Zhai, M. Tan, T. Xia and S. Wang. Multi-class Boosting with Direct 0-1 Loss Minimization and Margin Maximization. Technical Report, 2013.

[29] L. Zheng, S. Wang, Y. Liu and C. Lee. Information theoretic regularization for semi-supervised boosting. *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD), 1017-1026, 2009.

[30] J. Zhu, H. Zou, S. Rosset and T. Hastie. Multi-class AdaBoost. *Statistics and Its Interface*, 2:349–366, 2009.

## Appendix 1: Executive Summary of Lili Guo's Master Thesis

The main challenge in learning-to-rank for information retrieval is the difficulty to directly optimize ranking measures to automatically construct a ranking model from training data. It is mainly due to the fact that the ranking measures are determined by the order of ranked documents rather than the specific values of ranking model scores, thus they are non-convex, nondifferentiable and discontinuous. To address this issue, listwise approaches have been proposed where loss functions are defined either by exploiting a probabilistic model or by optimizing upper bounds or smoothed approximations of ranking measures. Even though very promising results have been achieved, there is still a mismatch between target cost and optimization cost. In this work, we present a novel learning algorithm that directly optimizes the ranking measures without resorting to any upper bounds or approximations. Our approach is essentially an iterative greedy coordinate descent method in optimization. For each iteration, we only update one parameter along one coordinate with all others fixed. Since the ranking measure is a stepwise function of a single parameter, we exploit an exhaustive line search algorithm to locate the interval with the smallest ranking measure along each coordinate. We pick the coordinate that leads to the largest reduction of ranking measure. In order to determine the optimal value of the parameter for the selected coordinate, we construct a probabilistic framework for the permutation, and maximize the likelihood of top-$m$ ranked documents. This iterative procedure is continued until convergence. We conduct experiments of five datasets selected from Microsoft LETOR datasets, our experimental results show that the proposed direct rank algorithm outperforms several well-known state-of-the-art ranking algorithms.